VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Parkovací asistent s laserovým skenerem

Parking Assistant with Laser Scanner

2019 Jakub Halman

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jakub Halman**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Parkovací asistent s laserovým skenerem
Parking Assistant with Laser Scanner

Jazyk vypracování: čeština

Zásady pro vypracování:

Využijte laserový skener, který slouží pro měření vzdálenost od překážek kolem sebe, pro řízení automatického zaparkování při podélném i kolmém stání. Pro řešení použijte minipočítač HummingBoard a LRF Hokuyo UST-20LX.

1. Seznamte se diplomovou prací zabývající se automatickým parkováním.
2. Seznamte se s platformou i.MX6 HummingBoard a laserový skenerem HOKUYO UST-20LX.
3. Seznamte se s modelem autem a jeho způsobem řízení.
4. Naprogramujte potřebné komunikační rozhraní pro laserový skener a řídící modul auta.
5. Navrhněte a sestavte program pro automatické zaparkování při podélném i kolmém státní.
6. Otestujte navržené řešení na různě velkých parkovacích místech.

Seznam doporučené odborné literatury:

[1] Automatické parkování automobilu, Luboš Matejčík, Diplomová práce, katedra informatiky, FEI, VŠB-TUO, 2017
[2] Laserový skener Hokuyo UST-20LX, https://www.hokuyo-aut.jp/
[3] Minipočítač HummingBoard: https://www.solid-run.com/nxp-family/hummingboard/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2019 .....

Rád bych na tomto místě poděkoval svému vedoucímu práce Ing. Petru Olivkovi, Ph.D. za cené rady a velmi užitečnou pomoc.

Abstrakt

Tato práce se zabývá návrhem parkovacího asistenta s použitím laserového skeneru na modelu auta postaveném na podvozku Alamak, který splňuje Ackermannovu podmínku. Laserový skener použitý v této práci je Hokuyo UST-20LX. Jako hlavní počítač byl použit počítač HummingBoard Pro od firmy SolidRun. Parkovacího asistenta se na navrženém řešení povedlo vytvořit jen částečně. Při řešení práce bylo objeveno několik technických problémů, které řešení komplikovaly.

Klíčová slova: Hokuyo UST-20LX, ICP, SLAM, Houghova transformace, HummingBoard, FRDM-K64F, FRDM-TFC, Ackermannova podmínka, Voxelizace

Abstract

This thesis deals with a design of parking assistent using laser scanner on a model of car, based on chassis Alamak, which meets Ackermann's condition. Laser scanner used in this thesis is Hokuyo UST-20LX. As a main computer was used a computer HummingBoard Pro manufactured by SolidRun. The parking assistant was developed only partially on the designed car model. Dealing with this thesis several technical problems were discovered, which complicated solution.

Key Words: Hokuyo UST-20LX, ICP, SLAM, Hough transform, HummingBoard, FRDM-K64F, FRDM-TFC, Ackermann condition, Voxel

Obsah

Seznam použitých zkratek a symbolů

ICP	–	Iterative Closest Point
SLAM	–	Simultaneous Localization And Mapping
FRDM	–	Freedom Development Platform
GPIO	–	General Purpose Input Output
USB	–	Universal Serial Bus
SoC	–	System on Chip
SPI	–	Serial Peripetal Interface
PCL	–	Point Cloud Library
CUDA	–	Compute Unified Device Architecture
FLANN	–	Fast Library for Approximate Nearest Neighbors
DC	–	Direct Curent
PWM	–	Pulse Wide Modulation
LRF	–	Laser range finder
LiDAR	–	Ligth Detection And Ranging
SOM	–	System On a Module
LVDS	–	Low-Voltage Differential Signaling
SSD	–	Solid-State Drive
Li-Pol	–	Lithium Polymer
LED	–	Light Emitting Diode
Mini-PCIe	–	Mini Peripheral Component Interconnect express
mSATA	–	mini Serial AT Attachment
μSD	–	Micro Secure Digital
SDA	–	Serial Data
SCL	–	Serial Clock
CMD	–	Command
CLK	–	Clock
GND	–	Ground
MISO	–	Master In Slave Out
MOSI	–	Master Out Slave In
TCP	–	Transmission Control Protocol
ARM	–	Advanced RISC Machine
IP	–	Internet Protocol
CSI	–	Camera Serial Interface
HDMI	–	High-Definition Multimedia Interface
S/PDIF	–	Sony/Philips Digital Interface
I^2C	–	Inter-Integrated Circuit

MCU – Micro Controller Unit

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Cílem mé práce je detekovat parkovací místo a následně na toto parkovací místo zaparkovat. Práce vychází z diplomové práce o Automatickém parkování automobilu od Luboše Matejčíka [1].

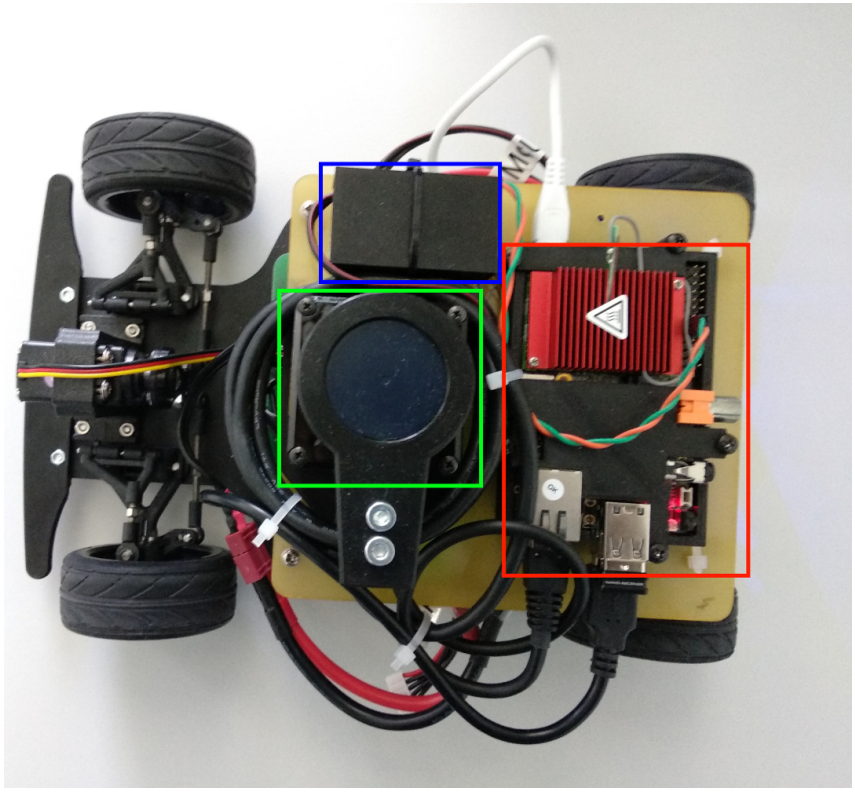
Nejdříve bude potřeba se seznámit s diplomovou práci o Automatickém parkování automobilu od Luboše Matejčíka [1]. Potom implementovat komunikační protokol použitého laserového skeneru, navrhnout třídu pro získávání dat ze skeneru a seznámit se s použitým modelem auta a způsobem jeho řízení. Následně se seznámit s možnostmi zpracování mračen bodů, týkajícími se převážně určování pozice a hledání určitých vzorů, jako jsou například úsečky a přímky. Poté by neměl být problém navrhnout řešení pro podélné a kolmé parkování a zjistit úspěšnost navrženého řešení.

Výsledkem práce by měl být program, který bude schopen vyhledat a rozlišit parkovací místo v mračnu bodů. A v případě nalezení vhodného parkovacího místa s modelem auta na toto místo autonomně zaparkovat.

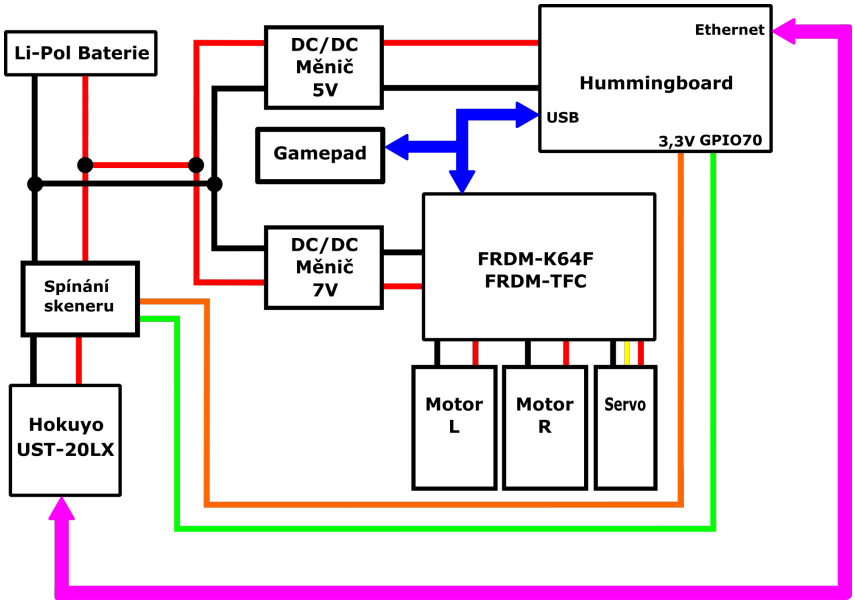
V dnešní době je již mnoho nových automobilů vybavených možností automatického zaparkování nebo nějakou formou parkovacího asistenta. Parkovací asistent se může například starat o natočení předních kol a řidič má na starost řízení rychlosti automobilu a brždění, podle požadavků automobilu. Pomocí těchto systému se zlepší parkování na parkovištích a díky tomu bude možné zaparkovat více aut na dané parkoviště z důvodu méně chybně zaparkovaných aut.

2 Model auta

Na obrázku 1 je vidět pohled shora na model auta. Červenou barvou je ohraničen hlavní počítač auta, který se stará o zpracování dat, zelenou barvou je ohraničen použitý laserový skener a modrou barvou je ohraničen obvod pro spínání napájení skeneru. Na obrázku 2 je vidět celkové blokové schéma zapojení celého modelu. Jednotlivé části budou popsány v následujících kapitolách.



Obrázek 1: Model auta



Obrázek 2: Blokové schéma zapojení

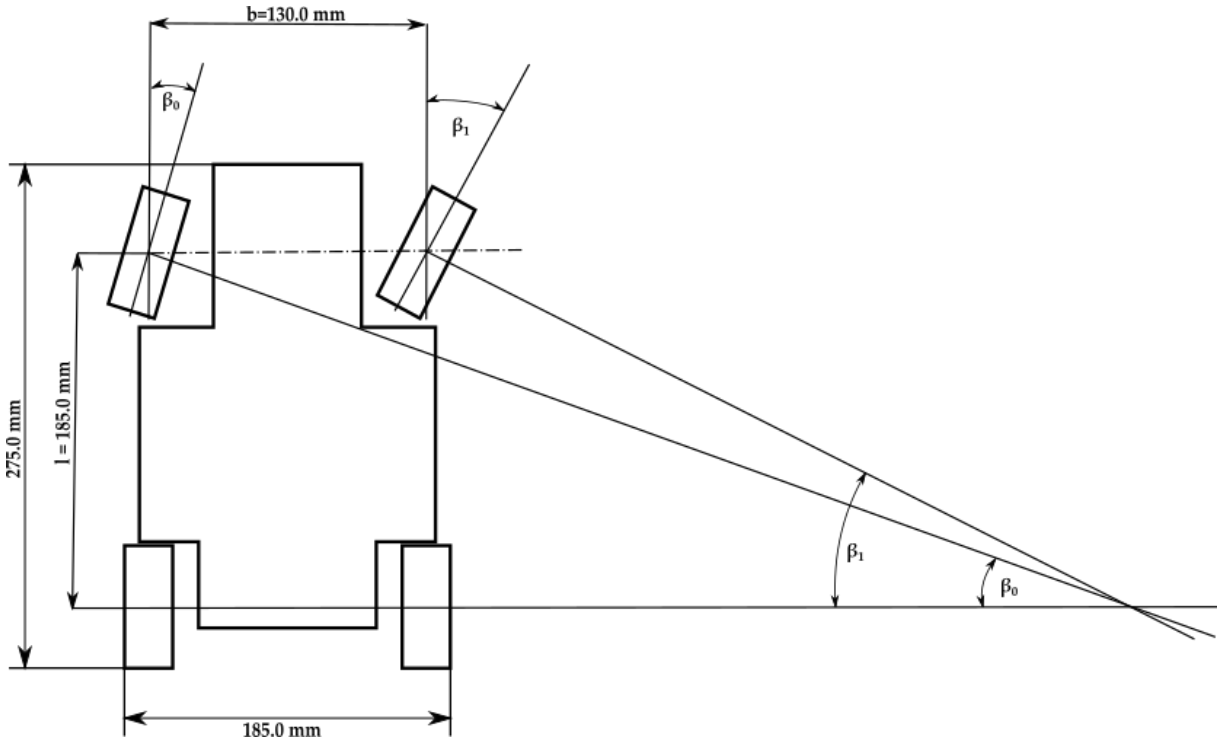
2.1 Podvozek

Pro podvozek modelu auta byl použit model Alamak, který je používán při závodech NXP Cup a splňuje Ackermannovu podmínku [9]. To znamená, že podvozek obsahuje čtyři kola, z toho dvě, která se natáčí a slouží k řízení směru. Natáčená kola mají každé mírně jiný úhel otočení. Dále je zadní náprava poháněna motory. Každé kolo zadní nápravy má v tomto případě vlastní motor, takže je možné při zatáčení upravovat výkon každého zadního kola zvlášť.

Ackermannova podmínka je dána vztahem:

$$cot\beta_0 - cot\beta_1 = -\frac{b}{l} \tag{1}$$

Kde β_0 je úhel vnějšího kola, β_1 je úhel kola vnitřního, b je vzdálenost mezi otočnými čepy a l je rozvor nápravy. Na obrázku 3 je nákres použitého podvozku modelu auta a jeho rozměry.



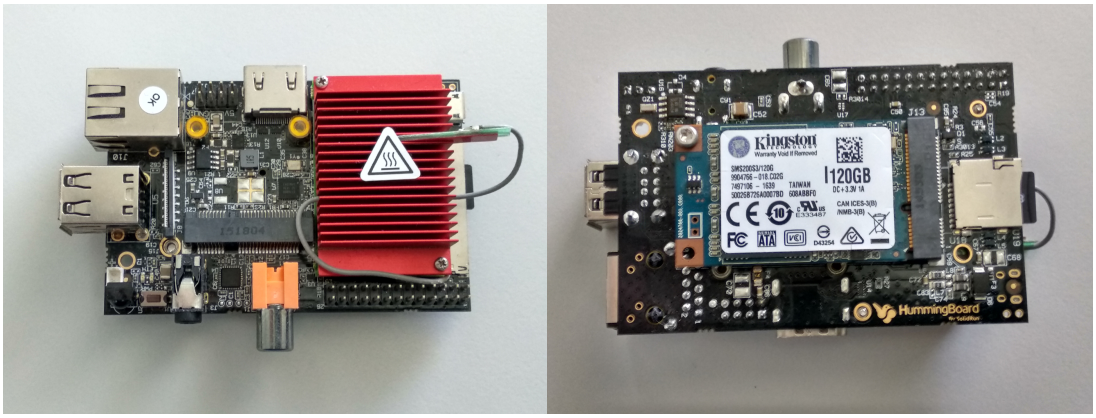
Obrázek 3: Nákres použitého modelu auta

2.2 Hlavní počítač

Jako hlavní počítač byl zvolen HummingBoard Pro od firmy SolidRun [13] viz obrázek 4, který má velikost jako počítač Raspberry PI, avšak oproti němu poskytuje řadu vylepšení a možností navíc. Najdeme zde čtyřikrát USB 2.0 (dvakrát jako USB A a dvakrát jen jako header), jeden HDMI konektor, S/PDIF výstup, 3,5 mm jack který slouží jako stereo výstup nebo jako mono vstup, Ethernetové rozhraní s rychlostí 1000 Mbps (jedná se o maximální rychlost linky, maximální přenosová šířka je 470 Mbps), 30pinový LVDS konektor, micro USB konektor sloužící k napájení zařízení, Mini-PCIe slot, slot pro μSD kartu, mSATA slot pro SSD, 15pinový konektor pro připojení kamery přes MIPI-CSI-2 a 26 GPIO pinů (pouze 8 pinů lze použít jako GPIO piny). V tabulce 1 se nachází popis funkce jednotlivých pinů.

HummingBoard je základní deska s i.MX6 SOM od firmy SolidRun. SOM na sobě obsahuje SoC i.MX6 [11] od firmy NXP, který může obsahovat až čtyři jádra ARM Cortex A9 s frekvencí až 1,2 GHz. Dále na SOM najdeme 2 GB operační paměti a Wi-Fi modul s podporovanými standardy 802.11a/b/g/n.

Jako paměťové uložiště byla použita μSD karta o kapacitě 8 GB a mSATA SSD disk s kapacitou 120 GB. Na μSD kartě se nachází pouze adresář /boot a zbylá adresářová struktura je na SSD, společně s 5 GB SWAP oddílem. Ve výpisu 1 z programu `lsblk`, lze vidět rozdělení disků na oddíly a jejich přípojné body.



Obrázek 4: Hummingboard Pro s SSD

Tabulka 1: Rozložení GPIO pinů

Signál	Pin	Pin	Signál
3,3 V	1	2	5 V
I2C_SDA	3	4	5 V
I2C_SCL	5	6	GND
GPIO 1	7	8	UART TX
GND	9	10	UART RX
GPIO 73	11	12	GPIO 72
GPIO 71	13	14	GND
GPIO 70	15	16	SD3_CMD
3,3V	17	18	SD3_CLK
SPI_MOSI	19	20	GND
SPI_MISO	21	22	GPIO 67
SPI_SCLK	23	24	ECSPI2_SS0
GND	25	26	ECSPI2_SS1

```
sda         8:0   0  111.8G  0 disk
-sda1       8:1   0  107G    0 part /
-sda2       8:2   0  4.8G    0 part [SWAP]
mmcblk0     179:0 0  7.4G    0 disk
-mmcblk0p1  179:1 0  7.4G    0 part /boot
```

Výpis 1: Rozdělení paměťových medií z programu `lsblk`

Do hlavního počítače je pomocí USB připojen bezdrátový gamepad a modul FRDM-K64F [12] s klonem modulu FRDM-TFC [20]. Dále je do hlavního počítače připojen laserový skener HOKUYO UST-20LX, a to pomocí Ethernetu. K zapínání skeneru se využívá jeden GPIO pin, přesněji pin 70.

Jako operační systém hlavního počítače byl zvolen Linux, přesněji Debian 8, bez grafického

rozhraní. V operačním systému bylo nutné upravit soubory Device Tree. Jedná se o binární soubory, které určují chování a nastavení různých periférií počítače, jako jsou GPIO piny a sběrnice např. I^2C, SPI a SATA.

Bylo zde potřeba upravit výchozí nastavení GPIO pinů. Tyto piny jsou po restartu počítače nastaveny jako vstupní s PULL-UP rezistorem, jenže po startu systému jsou přenastaveny na výstupní s hodnotou 0. To by způsobovalo zapínání a vypínání napájení skeneru, protože při reset stavu pinů trvá asi 5 sekund, než se systém načte.

Github společnosti SolidRun obsahuje repositář `linux-fslc`. V tomto repositáři je možné najít všechny soubory související z provozem Linuxu na jejich deskách, jako například i soubory dts a dtsi. Pro změnu defaultního nastavení GPIO pinů je důležitý soubor imx6qdl-hummingboard.dtsi. Přesněji se jedná o část s názvem `iomuxc` a o osm po sobě jdoucích řádků začínající řádkem `MX6QDL_PAD_GPIO_1__GPIO1_IO01 0x400130b1`. První část řádku je název GPIO pinu a druhá část je nastavení pinu. Hodnotu nastavení pinu je potřeba změnit na 0x4001b0b1. Poté jsem se pokusil zkompilovat tyto soubory pomocí programu dtc, což se nepovedlo.

Nakonec jsem se rozhodl původní Device Tree dekompilovat a dekompilovaný soubor upravit. Jediný problém takové úpravy je ten, že většina věcí je zde reprezentována adresou, ale stačí najít skupinu `hoggrp` a všechny hodnoty 0x400130b1 vyměnit za 0x4001b0b1. Nakonec stačilo takto upravený soubor znovu zkompilovat.

2.2.1 Instalace použitých knihoven

V práci byly použity dvě knihovny OpenCV [18] pro vykreslování dat na obrazovku a PCL [4] pro zpracování mračen bodů.

Jelikož se předpřipravené binární soubory knihovny OpenCV nachází ve standardních repositářích debianu i pro procesory ARM, tak nebylo nutné ji překládat ze zdrojového kódu. A také není nutné řešit závislost na jiných knihovnách. Instalace se provede zadáním jednoho příkazu: `apt-get install libopencv-dev`

Pro knihovnu PCL se předpřipravené binární soubory nenachází ve standardních repositářích debianu. Nachází se v repositáři launchpad (vyjma verze pro procesory ARM), a to ještě v zastaralé verzi. Jedinou možnost instalace knihovny byl tedy překlad ze zdrojového kódu. Výhodou překladu binárních souborů ze zdrojového kódu je možnost zvolit, co se bude překládat a co ne. Knihovna je například schopna využívat grafických karet NVIDIA s architekturou CUDA (Compute Unified Device Architecture). Jelikož hlavní počítač neobsahuje grafickou kartu NVIDIA, je tento modul knihovny zbytečný a zkrátí čas překladu. Dále jsem nepřekládal modul visualisation, protože pro vykreslování jsem použil knihovnu OpenCV, která nabízí možnost vykreslování podle potřeby. Nevyužitím tohoto modulu se odstranila závislost na knihovně VTK. Knihovna PCL pro svou funkci dále potřebuje ještě knihovny Boost [16] (soubor knihoven, který nabízí podporu práce v oblasti lineární algebry, zpracování obrazu, více vláknových operacích a mnoha dalších, obsahuje přes 80 knihoven), Eigen [14] (práce s lineární algebrou, maticemi

a vektory) a FLANN [15] (Fast Library for Approximate Nearest Neighbors, podpora pro vyhledávání sousedních bodů v prostoru). I tyto knihovny jsem se rozhodl přeložit ze zdrojového kódu, aby byly splněny podmínky minimální verze knihovny.

Instalace knihovny Boost proběhla podle návodu. Po stažení a rozbalení se spuštěním shell skriptu `./boostrap.sh` vytvoří instalační program. Pro instalaci stačí tento program spustit příkazem `./b2 install`.

K překladu knihovny Eigen se používá nástroj CMake. CMake je program pro multiplatformní překlad zdrojového kódu pod různými operačními systémy. Pro kompilaci pomocí nástroje CMake je potřeba mít vytvořený soubor CMakeLists.txt. Tento soubor tak obsahuje všechny soubory a nastavení, které se mají použít při překladu. Výsledkem nástroje CMake je pod unixovými systémy soubor Makefile. Instalace knihovny Eigen při použití programu CMake je následující: nejprve je vhodné ve složce s binárními soubory knihovny vytvořit složku build a přejít do této složky. Ve složce se zadá příkaz `cmake ..`, který říká ať program CMake hledá soubor CMakeLists.txt o sloužku výše. Poté se ve složce build vytvoří soubor Makefile. Příkazem `make install` se spustí překlad a následná instalace knihovny.

Překlad knihovny FLANN probíhá, stejně jako v případě instalace knihovny Eigen, pomocí CMake, tedy i postup je stejný.

Pro překlad knihovny PCL se taky používá program CMake. Při překladu jsem ale nepoužil přímo program CMake, ale program ccmake. Ten obsahuje navíc grafickou nadstavbu, takže je možné jednoduše zakázat, které moduly chceme přeložit a které ne. Navíc je vhodné přidat parametr `DCMAKE_BUILD_TYPE` nastavený na hodnotu `Release`, čímž jsou zapnuty optimalizace překladače. Dále je potřeba u příkazu `make` nastavit úroveň optimalizace. Pro testování je doporučena co nejmenší optimalizace z důvodu rychlejšího překladu kódu. Po přeložení je nutné už jen zadat příkaz, který knihovnu nainstaluje. Výsledná posloupnost příkazů bude vypadat podobně jako ve výpisu 2.

```
ccmake -DCMAKE_BUILD_TYPE=Release ..
make -j2
make -j2 instal
```

Výpis 2: Posloupnost příkazů pro překlad a instalaci knihovny PCL

Celková doba překladu použitých knihoven byla něco kolem 12 hodin.

2.2.2 Ovládání GPIO pinů

V operačních systémech Linux je každé zařízení reprezentováno jako soubor v adresářové struktuře většinou ve složce `/dev`, soubory GPIO pinů se však nachází ve složce `/sys/class/gpio`.

Pokud chceme pracovat s některým z GPIO pinů, je nutné jej nejprve vyexportovat. To je možné realizovat zapsáním jeho čísla do souboru `export`. To se dá provést příkazem, kde místo `XX` je číslo GPIO pinu:

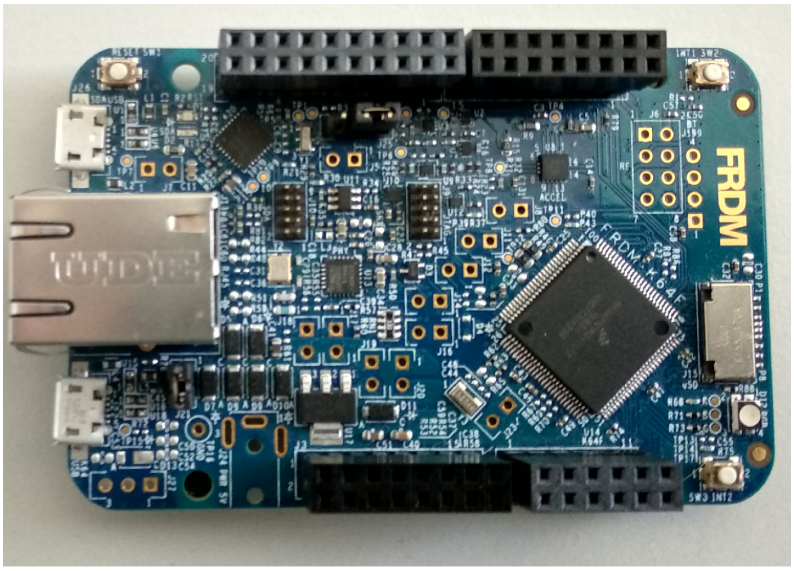
`echoXX > /sys/class/gpio/export`

Potom by v adresáři měl být vidět adresář `gpioXX`. V tomto adresáři se nachází soubory pro práci s GPIO pinem. Pro nastavení, jestli bude pin vstupní nebo výstupní, slouží soubor `direction`. Pokud má být pin nastaven jako vstupní, stačí do souboru zapsat `in` a pokud jako výstupní, tak `out`. Pokud je potřeba zjistit logickou hodnotu vstupního pinu, který je nastavený jako vstupní, stačí přečíst hodnotu v souboru `value`. Tento soubor slouží také k nastavení hodnoty výstupního pinu. Hodnoty se v souboru zobrazují jako číslice jedna a nula.

Tímto způsobem nelze u GPIO pinů nastavovat jejich PULL-UP, PULL-DOWN rezistory.

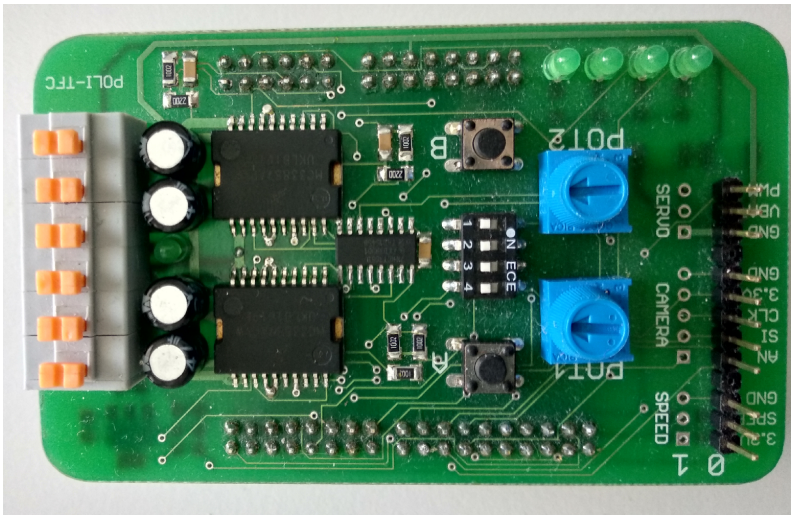
2.3 FRDM-K64F a klon FRDM-TFC

Pro ovládání motorů je použit modul FRDM-K64F [12], který je zobrazen na obrázku 5, od firmy NXP (dále jen K64F), na který je nasazen klon rozšiřujícího modulu FRDM-TFC [20] viz obrázek 6. K64F je vývojová platforma pro Kinetis®. K64F je postaven na mikropočítači MK64FN1M0VLL12 (dále jen MCU) od firmy NXP. MCU je založeno na ARM Cortex M4, běží na frekvenci 120 MHz, datová paměť má velikost 256 kB a programová má velikosti 1 MB. K64F obsahuje dva micro USB porty, jeden je určen pro programování a druhý je připojen přímo k MCU a je možné jej naprogramovat dle potřeby (v této situaci simuluje sériovou linku), dále zde nalezneme slot pro μSD kartu a konektor RJ45 pro připojení do sítě Ethernet.



Obrázek 5: FRDM-K64F

Klon FRDM-TFC (dále jen TFC) je rozšiřující modul, který obsahuje dva H-můstky MC33887 sloužící pro řízení dvou DC motorů. Můstky jsou vybaveny feedback výstupem pro měření proudu proudícího do motoru. Tento proud je potom čten pomocí AD převodníku MCU. Prostřednictvím TFC je možné dále řídit dva servomotory pomocí PWM a také číst data ze dvou řádkových kamer a dvou snímačů otáček. Dále zde nalezneme čtyři LED diody, dvě tlačítka a čtyři DIP přepínače pro uživatelské nastavení.



Obrázek 6: Klon FRDM-TFC

Komunikace mezi hlavním počítačem a K64F probíhá prostřednictvím virtuálního sériového kanálu, který vytváří K64F. Přes tento kanál jsou posílány tři různé datové struktury. Dvě slouží k přenosu dat z hlavního počítače a jedna pro přenos dat z K64F. Pro komunikaci s TFC jsou použity knihovny `TFC` a `TFCSerial` z diplomové práce o Automatickém parkování automobilu [1].

2.4 Napájení modelu

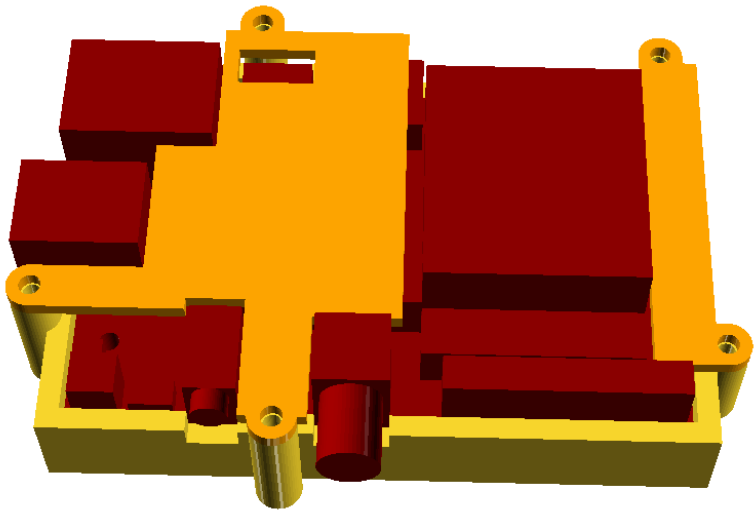
Pro napájení všech komponent modelu auta je potřeba tří různých napětí. Je potřeba napětí od 10 do 30 V pro laserový skener, 7 V pro napájení motorů podvozku a 5 V pro napájení hlavního počítače. Celé zapojení napájení se nachází na obrázku 2.

Jako napájecí zdroj byl použit akumulátor typu Li-Pol s kapacitou 5000 mAh, který má nominální napětí 14,8 V a napětí na prázdno při plném nabití je 16,8 V. Laserový skener je připojen přímo k baterii. Pro napájení motorů je použit DC/DC měnič který snižuje napětí dodávané baterií na 7 V, pro napájení hlavního počítače je použit stejný měnič jen s výstupním napětím nastaveným na 5 V.

2.5 Uchycení jednotlivých zařízení

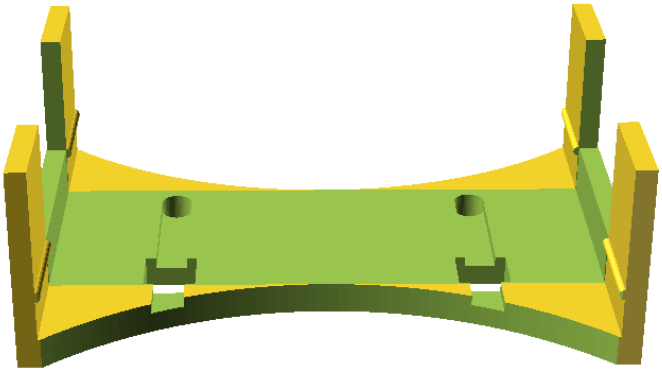
Pro většinu použitých zařízení byly v programu OpenSCAD vymodelovány držáky. Tyto držáky byly poté vytisknuty na 3D tiskárně.

Nejdůležitějším, ale také nejsložitějším, je držák pro uchycení hlavního počítače. Hummingboard totiž obsahuje pouze dva montážní otvory. Z toho jeden je kompletně zakryt modulem SOM a druhý z důvodu rozmístění okolích součástek také není použitelný. Z těchto důvodů byla vytvořena krabička pro hlavní počítač. Tato krabička se skládá ze tří částí: jedné spodní části, do které je zasezen hlavní počítač, a dvou horních dílu, které drží hlavní počítač ve spodním dílu tak, aby nevypadl. Horní díly jsou ke spodnímu dílu připevněny pomocí pěti šroubů. Na obrázku 7 je pomocí červené barvy označen model počítače HummingBoard Pro, žlutou barvou je označena spodní část krabičky a oranžovou barvou jsou označeny dva horní díly, kterými je pročítač držen na místě.



Obrázek 7: Model držáku pro počítač HummingBoard Pro

Většina držáků je připevněna ke sklotextitové desce, která je přišroubována k podvozku modelu auta. Držák pro FRDM-K64F je pomocí stahovacích pásků připevněn k podvozku auta. Model držáku je zobrazen na obrázku 8.



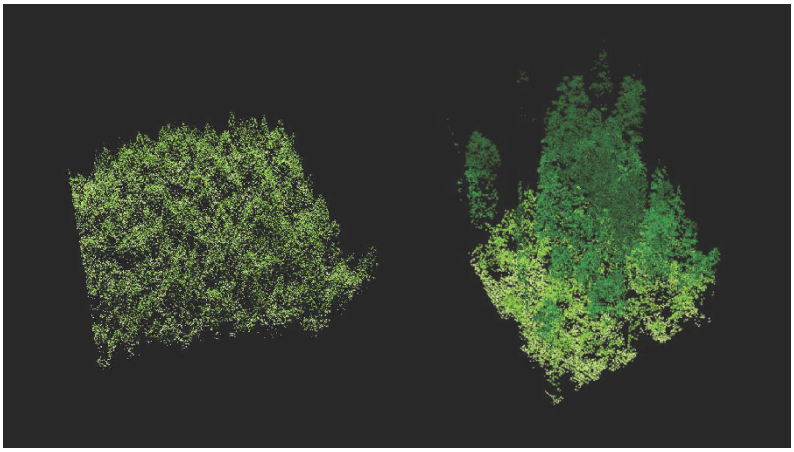
Obrázek 8: Model držáku pro FRDM-K64F

3 Laser Rangefinder

Laser Rangefinder, zkráceně LRF, je zařízení využívající laserový paprsek k měření vzdáleností od překážek na základě doby síření pulsu laserového paprsku. Jedná se o metodu LiDAR. Z důvodu vysoké rychlosti světla není LRF vhodné pro měření s přesností menší než milimetr [5].

3.1 Využití metody LiDAR

LiDAR (Light Detection and Ranging) [6] je převážně využíván k tvorbě map s vysokým rozlišením a lze ho využít i k lokalizaci v prostoru. Metoda je využívána například k tvorbě výškových map, kdy je skener umístěn na letadle nebo satelitu letícím nad určitou oblastí, v archeologii ke snímání archeologických míst, k laserovému navádění nebo na autonomních autech a robotech k detekci překážek a orientaci v prostoru. Na obrázku 9 jsou zobrazeny dva skeny stejného lesa s odstupem času.



Obrázek 9: Příklad mračna bodů získaná metodou LiDAR [6]

3.2 Princip laserového skeneru

Laserový skener se skládá ze tří hlavních částí, a to ze zdroje laserového paprsku, z mechanické části, která slouží k nastavení snímaného bodu a ze senzoru, který přijímá odražený laserový paprsek.

Jako zdroj laserového paprsku se nejčastěji využívá laserová LED dioda. Pro nevědecké účely je využíván laser s vlnovou délkou od 600 nm do 1000 nm, maximální výkon laseru musí být limitován nebo při výškových měřeních vypínán pod určitou výškou, aby nedošlo k poškození očí. Alternativou je laser s vlnovou délkou 1550 nm, problémem této vlnové délky je však detektor, který není dostatečně přesný. Tato vlnová délka se používá hlavně u armádních zařízení, protože ji nelze spatřit pouhým okem, ani pomocí brýlí na noční vidění. Pro topografické mapování se nejčastěji využívá vlnová délka 1064 nm a pro podmořské mapování vlnová délka 532 nm, u které dochází k nižšímu útlumu při průchodu vodou než u vlnové délky 1064 nm.

Mechanická část je většinou tvořena zrcátkem umístěným na krokovém motoru. Zrcátko vytváří odraz snímaného místa, takže laserový paprsek je odražen zrcátkem a odraz laserového paprsku taktéž. Senzor a zdroj laserového paprsku je umístěn nad nebo pod zrcátkem podle tipu laserového skeneru. Otáčením zrcátka je snímána jiná část prostoru a tak není potřeba složitého přenosu dat ze senzoru. Zrcátko se může natáčet podle jedné nebo dvou os v souvislosti s tím, jestli se jedná o 2D skener nebo 3D skener.

Jako senzor je nejčastěji využívána fotodioda pro vlnovou délku laserového paprsku. Laserový paprsek může být modulován, aby bylo těžší rušit měření skenerem.

Výpočet vzdálenosti, změřené skenerem, je dán vztahem

$$D = \frac{c.t}{2} \tag{2}$$

Kde c je rychlost světla a t je doba od vyslání paprsku po jeho odražení zpět. Tuto hodnotu je potřeba vydělit dvěma, protože paprsek putuje tam a zpět, takže trasu urazí dvakrát.

3.3 Hokuyo UST-20LX

V této práci byl použit skener Hokuyo UST-20LX [17], který je vyfocen na obrázku 10. Skener vyžaduje napájení v rozmezí 10 až 30 V, není tedy problém jej přímo připojit k baterii auta. Skener je k baterii připojen přes MOSFET typu P, který je ovládán prostřednictvím GPIO pinu 70 přes emitor NPN tranzistoru. Skener využívá laser o vlnové délce 905 nm, a patří do laserové třídy 1, takže toto zařízení nepoškodí oči ani při pozorování pomocí očních pomůcek [22], jako jsou např. brýle, lupy. Rozsah detekce je od 0,06 m do 20 m při použití bílého listu a od 0,06 m do 8 m při 10% difúzní odrazivosti. Snímaný úhel je 270° s úhlovým rozlišením 0,25°. To znamená, že pro jeden celý sken je pořízeno 1081 hodnot.

Pro komunikaci s hlavním počítačem skener využívá protokol SCIP [19] přenášený pomocí transportního protokolu TCP na portu 10940. Port nelze změnit.

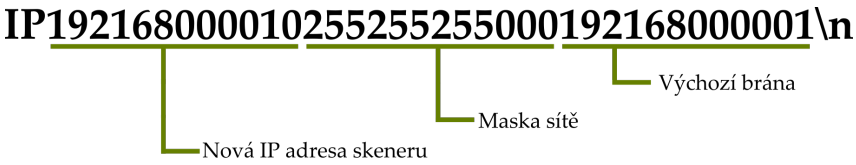
Obrázek 10: Hokuyo UST-20LX

3.3.1 Změna IP adresy

Skener má výchozí nastavenou IP adresu 192.168.0.10 s maskou 255.255.255.0 a výchozí bránu na 192.168.0.1. Pro změnu slouží příkaz `IP`. IP adresa, maska i adresa brány se zapisuju znakově. Každých 8 bitů je reprezentováno pomocí tří znaků čísel, takže je nutné nepoužité místo nahradit nulou, např. 255.255.0.0 bude vypadat 255255000000.

Pro změnu IP adresy je nutné zařízení okamžitě restartovat příkazem `RB\nRB\n`. Po restartu by měl mít skener již novou IP adresu. To se dá vyzkoušet jednoduše pomocí aplikace telnet. Pokud se spojení na daném portu pod novou adresou povede, adresa zařízení byla úspěšně změněna.

Při změnách IP adresy se stávalo, že se adresa nezměnila hned na poprvé. Stačilo ale stejnou posloupnost príkazů poslat vícekrát a IP adresa skeneru se změnila.



Obrázek 11: Popis příkazu IP

Na obrázku 11 lze vidět příklad příkazu IP který nastaví IP adresu, masku a bránu na výchozí hodnoty. Po každém příkazu odešle skener kopii příkazu a status hodnotu.

3.3.2 Čtení dat ze skeneru

Existují čtyři příkazy pro získání pouze naměřených vzdáleností ze skeneru `MD`, `MS`, `GD` a `GS`. Příkazy začínající na M se používají, pokud chceme data získávat kontinuálně. Takže stačí jednou

poslat příkaz a data nám budou přicházet, dokud neuzavřeme TCP kanál, nepošleme příkaz QT nebo dokud nepřijmeme požadovaný počet skenů (pokud tedy nebylo nastaven na nekonečno). Zatímco příkazy začínající na G slouží pro získání jednoho skenu, před tímto příkazem je nutné ještě odeslat příkaz BM, kterým se zapne laserová dioda. Příkazy končící na D posílají data v tříbajtovém formátu (maximální vzdálenost kterou lze takhle přenést je 262144 mm) a příkazy končící na S posílají data v dvoubajtovém formátu (maximální vzdálenost kterou lze takhle přenést je 4085 mm). Pro dekódování dat (vzdálenost, časové razítko, intenzita) získaných ze skeneru lze použít následují funkci ve výpisu 3

```
int Scanner::Decode(const char code[], int byte)
{
   int value = 0;
   for (int i = 0; i < byte; ++i)
   {
      value <<= 6;
      value &= ~0x3f;
      value |= code[i] - 0x30;
   }
   return value;
}
```

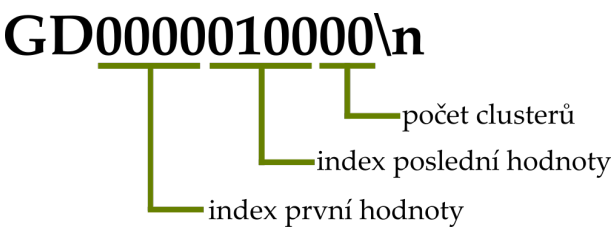
Výpis 3: Funkce pro dekódování dat ze skeneru [3]

Při použití příkazu GD nebo GS pro získání aktuálních naměřených hodnot ze skeneru, nejdříve odpoví kopií příkazu a status kódem. Poté následuje časové razítko skenu. Hodnota časového razítka se odvíjí od hodnoty vnitřního 24bitového čítače ve skeneru. Poté již následují samotná data. Data přichází v řádcích po 65 znacích (bajtech) plus ukončovací znak (LF). Toto neplatí pro poslední řádek dat, kde se délka odvíjí od počtu zbývajících dat, které ještě nebyly odeslány. Prvních 64 znaků jsou zakódované hodnoty a 65. znak je kontrolní součet. Při použití 3bajtového přenosu (příkazy GD a MD), může být na konci řádku jedna neukončená hodnota. Bylo tedy potřeba, aby byl použit bajt z nového řádku, a ne kontrolní součet nebo ukončovací znak. Když skener odešle poslední řádek dat, odešle po něm ještě jeden prázdný řádek.

Po použití příkazu GD dle obrázku 12, například přes program telnet, dostaneme ve výsledku ze skeneru například tato data:

```
GD0000010000
00P
1<7Xl
0>J0>B0>B0>C0>J0>M0>N0>40>40>K0>P0>a0>Z0>m0?00?70?70?30?L0?G0@>12
HV1HR1HQ1HP1H31GQ1GP1Hd1H:1EE1EE1K[1Kh1LV1Li1Mb1P31PQ1PM1PO1PK1P4
G1Pk1R71Se1UK1Vj1WQ1W\1Wh1Y@1YT1YO2K@2K@2K>2K52Jj2JB2JG2JO2Io2I1F
2I92IG2IN2IB2I82He2Hk2HQ2HK2HK2HE2Gm2Gj2Gn2GR2GE2GE2G?2G62Fl2Fh2K
```

```
Fi2Fa2FK2FM2FE2FJ2FL2Em1U@1U>1U>1UI1UI1UF1UH1UG:
```



Obrázek 12: Popis příkazu `GD`

Popis parametrů příkazu `GD` z obrázku 12:

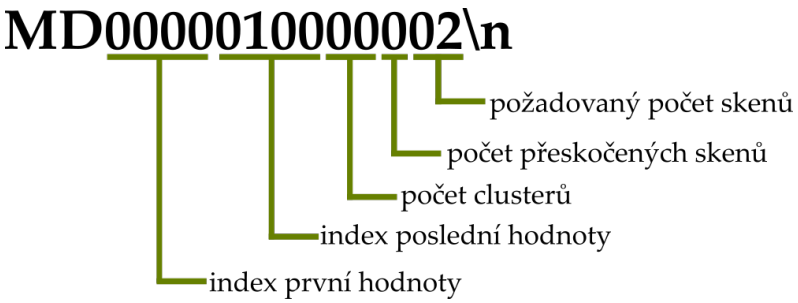
- Index první hodnoty - Určuje od kterého indexu, neboli úhlu, jsou data požadována. Pokud je hodnota 0, tak úhel od kterého se začíná je 270°.
- Index poslední hodnoty - Určuje který index bude započítán jako poslední, neboli který úhel se započítá jako poslední, v tomto případě 295°.
- Počet clusterů - Určuje kolik naměřených vzdáleností bude přeskočeno, v tomhle případě žádné.

Pokud je pro čtení dat využito příkazu `MD` nebo `MS`, tak nejprve skener pošle kopii příkazu a po něm status, stejně jako u příkazu `GD`. Po těchto dvou řádcích následuje prázdný řádek a kopie příkazu kde je místo počtu požadovaných skenů počet skenů, které po tomto skenu ještě přijdou. Po té následuje status, časové razítko a data. Ta jsou zde formátována stejně jako u příkazů `GD` a `GS`.

Po použití příkazu na obrázku 13, bude odpověď ze skeneru vypadat nějak takto:

```
MD0000010000002
00P

MD0000010000001
99b
0M07T
0>D0>J0>;0>?0>40>70>:0>C0>C0>30>:0>=0>K0>I0>A0>E0>B0>A0>B0>V0>_0W
>R0>Y0?20?40?30>o0?30?40?U1F>1EN1Jn1Lf1Li1MV1OK1PZ1PÎPo1PQ1PW1P2
K1Pa1RV1Sf1UV1VY1WM1WR1Wh1YA1YÎYU2K=2K72K22J‘2Je2JP2JE2J92Im2J6C
2Im2I\2IC2I22He2Hi2HÎHY2HQ2HT2HL2H=2Ga2GW2Gk2GZ2GG2GI2G62Fi2F_2Y
G;2G52G42F_2FS2FK2FC2F;1UW1U:1UB1UE1UJ1UI1UM1UAI

MD0000010000000
99b
```

```
OMOPm
O>CO>HO>@O>2O>2O>EO>EO>EO>6O>9O>?O>GO>GO>JO>QO>MO>GO>GO>KO>MO>TOB
>PO>gO>hO?AO?=O?6O?9O?AO?61F41Fj1KD1L‘1Ll1Md1OT1Pe1P‘1PU1PI1PJ1Pb
H1Pc1RK1So1UB1Ve1WH1Wi1Wl1Y@1YL1ZQ2KF2K>2K52Jd2Jb2JS2JA2J:2J12J1L
2Ii2IW2IG2I:2Hn2He2HT2HR2HV2HI2HD2H12GĜGk2Ga2GR2GO2G42Fh2Fh2Fi2b
G92Fe2FT2FP2FP2FP2FN2FN1Ul1U41U21UL1UK1UB1UC1UCˆ
```



Obrázek 13: Popis příkazu `MD`

Popis paramtrů příkazu MD z obrázku 13:

- Index první hodnoty, Index poslední hodnoty, Počet clusterů - Tyto parametry zůstavají stejné jako u příkazu `GD`.
- Počet přeskočených skenů - Určuje kolik skenů, se má přeskočit mezi každým odeslaným skenem.
- Požadovaný počet skenů - Určuje kolik skenů je požadováno po skeneru. Pokud je zadána hodnota 00, tak bude počet skenů nekonečný.

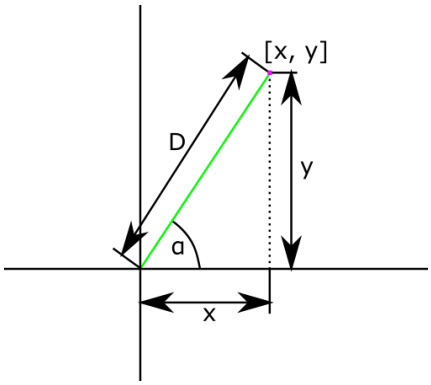
3.4 Převod vzdáleností na mračno bodů

Skener vrací vzdálenost skeneru od překážky. Tuto hodnotu je nutné převést na mračno bodů. Mračno bodů je většinou datové pole, které obsahuje datové struktury bodů, přičemž každý bod má svojí polohu reprezentovanou souřadnicemi (podle typu mračna jsou tyto souřadnice buď trojrozměrné, nebo dvourozměrné). Dále může být k bodu uvedena jeho barva, intenzita a jiné hodnoty.

Skener použitý v této práci je pouze dvourozměrný, takže nemáme údaj o výšce. K přepočtu úhlu a vzdálenosti na souřadnice se využívá těchto rovnic:

$$\begin{aligned} x_i &= cos\alpha_i * D_i \\ y_i &= sin\alpha_i * D_i \end{aligned} \tag{3}$$

Kde D je naměřená vzdálenost od překážky a α je úhel ve kterém byla vzdálenost změřena. Jedná se o převod polárních souřadnic na kartézskou soustavu souřadnic. Na obrázku 14 lze vidět rozdíl mezi polární a kartézskou soustavou souřadnic [21].



Obrázek 14: Převod polární soustavy souřadnic na kartézskou soustavu souřadnic

4 Odstranění potenciálně chybných bodů v mračnu bodů

Při měření laserovým skenerem se můžou v naměřených datech vyskytovat chybná měření, která nepopisují skutečnou situaci. Tyto body potom ztěžují nebo dokonce znemožňují práci dalších algoritmů pracujících nad daným mračnem bodů a ty potom poskytují nesprávné výsledky. Chybové body vznikají převážně při přechodu přes ostrou hranu nebo následkem nevhodných vlastností materiálu.

Po prozkoumání několika skenů obsahujících chybné body jsem zjistil, že takovéto body se většinou vyskytují osamoceně, čili nemají žádné, nebo skoro žádné blízké body. Toho jsem se rozhodl využít při odstraňování těchto bodů, takže v případě, že bod nemá určitý počet sousedů ve svém okolí, je považován za chybný a z mračna je odstraněn.

Pro vyhledání nejbližších bodů jsem v práci využil k-dimenzionálního stromu (k-d tree). K-dimenzionální strom má průměrnou časovou složitost pří hledání O(log n) a maximální složitost O(n). Jedná se o binární strom, kde každý uzel, který je list (čili nemá žádné potomky) je bodem v k-dimenzionálním prostoru. Pokud uzel není listem, tak rozděluje prostor na dvě části. Každému takovému uzlu je přiřazena jedna z k-dimenzí. Všechny body, které mají hodnotu v dané dimenzi menší, jsou zařazeny do levého podstromu, a všechny větší do pravého podstromu.

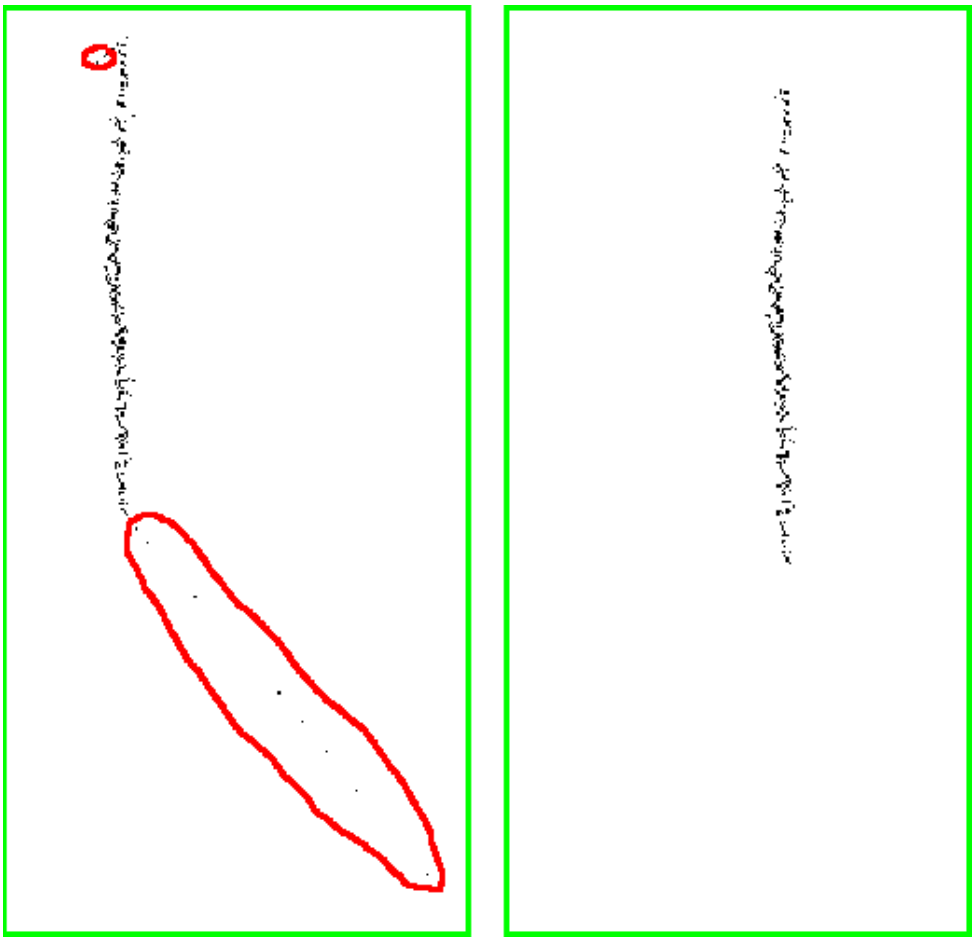
```
pcl::PointCloud<pcl::PointXYZ>::Ptr Scanner::ShadowPointsRemove
(pcl::PointCloud<pcl::PointXYZ>::Ptr cloud)
{
   const double max_points_distance = 0.015 //Velikost okolí bodu

   pcl::PointCloud<pcl::PointXYZ>::Ptr ret(new pcl::PointCloud<pcl::PointXYZ>);
   ret->height = cloud->height;
   ret->width = cloud->width;
   ret->points.resize(ret->height * ret->width);
   pcl::search::KdTree< pcl::PointXYZ> tree;
   tree.setInputCloud(cloud);

   std::vector<int> indices;
   std::vector<float> distance;

   int index = 0;

   for (int i = 0; i < cloud->points.size(); i++)
   {
      if (tree.radiusSearch(i, max_points_distance, indices, distance) > 5)
      {
```

```
            ret->points[index] = cloud->points[i];
            index++;
        }
    }

    if (index < ret->points.size())
    {
        ret->width = index;
        ret->points.resize(ret->height * ret->width);
    }

    return ret;
}
```

Výpis 4: Funkce pro odstranění chybných bodů

Nevýhodou řešení z výpisu 4 je, že pokud chybových bodů bude hodně u sebe z důvodu například vysoké odrazivosti materiálu, tak je toto řešení bude považovat za body správné a ponechá je. Další nevýhodou tohoto řešení je, že na větší vzdálenosti od skeneru se vzdálenosti i mezi body zvětšují. To způsobí, že nemusí být nalezen dostatečný počet sousedních bodů v daném okolí a bod bude považován za chybný a odstraněn.

Na obrázku 15 lze vidět použití filtru. Skoro všechny body, které vznikly na přechodu přes hranu byly odstraněny, takže nebudou ovlivňovat další zpracovaní mračna. Algoritmus vyhledává sousední body v okolí 15 mm, přičemž těchto sousedních bodů musí mít bod více než 5, jinak je považován za chybný a odstraněn.



(a) Před použitím filtru (b) Po použitím filtru

Obrázek 15: Před a po použití filtru

5 Lokalizace a mapování

Pro správné zaparkování je nutné, aby auto vědělo, kde se nachází a kde se nachází parkovací místo. Tomuto problému se říká SLAM (Simultánní Lokalizace a Mapování) [8]. Jelikož se nejedná o auto s autonomním řízením, tak není potřeba ukládat mapu celé trasy. Stačí mít uloženou pouze mapu nebližšího okolí obsahujícího parkovací místo.

K tomu byl v této práci zvolen algoritmus ICP [7] a po určitém posunu se provede voxelizace mapy, aby se odstranily duplicitní body a zároveň aby se zmenšila paměťová náročnost mapy. Pokud auto neparkuje, tak se ještě provádí posun celé mapy tak, aby pozice auta byla nulová a úhel natočení auta byl 90° (úhel otočení o osu z 0°) a nad takto upravenou mapou se provede filtrování bodů. Ty, které nepatří do určitého rozsahu, jsou odstraněny, čímž se zajistí, že mapa obsahuje jen nejbližší okolí auta.

5.1 Iterative Closest Point

Iterative Closest Point, dále jen ICP, je algoritmus sloužící k minimalizaci vzdálenosti mezi dvěma mračny bodů.

Vstupními parametry ICP algoritmu je zdrojové mračno bodů (mračno které má být transformováno na jiné mračno), cílové mračno bodů (mračno na které má být zdrojové mračno transformováno), předpokládaný posun mračna (tento posun je buď určen pomocí jiného algoritmu, nebo pomocí měření otáček kol) a kritéria pro zastavení algoritmu. V použitém variantě ICP jsou kritéria takováto: maximální počet iterací algoritmu (pokud je tento počet překročen, algoritmus je ukončen), transformační epsilon (pokud je rozdíl mezi současnou a předešlou transformací menší než epsilon algoritmus je ukončen) a Euklidovské epsilon (pokud je součet chyb Euklidovské vzdálenosti menší než zadané epsilon, algoritmus končí). Výstupem algoritmu je transformační matice. Průběh ICP algoritmu je následující:

- pro každý bod ve zdrojovém mračnu se vyhledají korespondenční body nebo bod v cílovém mračnu, nejčastěji se hledají body, které jsou si nejblíže, nebo mají podobné vlastnosti (intensita, barva),
- odstranění nevhodných korespondencí, nemusí být využito,
- odhadnutí transformace a transformace zdrojového mračna,
- pokud nebylo dosaženo zadaných kritérií, tak proběhne nová iterace s použitím odhadnuté transformace.

5.1.1 Nalezení korespondenčních bodů

Existuje několik způsobu nalezení korespondenčních, mezi nečastější patří point to point a point to plane.

Point to point neboli bod na bod, je asi nejjednodušší metoda pro vyhledávání korespondenčních bodů. K bodu ve zdrojovém mračnu se vyhledá nejbližší bod v cílovém mračnu. Pokud vzdálenost mezi těmito body není větší, než zvolená maximální vzdálenost korespondenčních bodů, je považována za vhodný korespondenční bod. Problémem takovéhoto určování korespondenčních bodů je velké ovlivnění korespondenčními body, které mají velmi nízkou či nulovou vzdálenost mezi sebou, což způsobuje nesprávné transformace, které zkreslují či znemožňují mapování a určení pozice. K vyhledávání nejbližších sousedních bodů lze využít k-dimenzionálního stromu.

Point to plane, neboli bod na rovinu, je metoda, jež nelze moc dobře využít na 2D mračnech, protože rovina v dvourozměrném prostoru neexistuje. Ekvivalentem roviny ve dvourozměrném prostoru je přímka. Ke každému bodu je určena rovina a jejich normálový vektor, pomocí kterých je pak nalezen ideální bod v cílovém mračnu bodů. Nejčastěji se k tomu používá Levenberg-Marquardtova metoda.

V této práci byla využita převážně metoda point to point, dále jsem ještě zkoušel vyhledávat korespondenční body pomocí nalezení úseček v mračnech bodů. Úsečka je definována dvěma body. Tyto body byly potom využity k hledání korespondenčních bodů. Průběh algoritmu vypadal takto:

- Nalezení úseček ve zdrojovém mračnu bodů a nalezení odpovídajících bodů ve zdrojovém mračnu k bodům definujícím úsečku.
- Nalezení úseček v cílovém mračnu bodů a nalezení odpovídajících bodů v cílovém mračnu k bodům definujícím úsečku.
- Ke všem nalezeným bodům ze zdrojového mračna se najde nejbližší bod mezi nalezenými body z cílového mračna.

5.1.2 Odstranění nevhodných korespondenčních bodů

Při určování korespondencí se může stát, že některé korespondence jsou vybrány chybně a mohou nevhodně ovlivnit práci ICP algoritmu.

Jedním z nejčastějších problémů při určování korespondencí je skutečnost, že několik bodů ze zdrojového mračna je v korespondenci se stejným bodem v cílovém mračnu. To může způsobit problémy při hledání vhodné transformace.

Některé filtry pro odstranění nevhodných korespondencí využívají vlastnosti jako je barva nebo intenzita bodu.

V této práci jsem využil filtr, který nejdříve zjistí průměrnou vzdálenost korespondenčních bodů a potom odstraní všechny, které mají menší vzdálenost, než je ta průměrná. Tím jsou odfiltrovány korespondence, které mají nulovou nebo hodně malou vzdálenost a způsobovaly nesprávné rotace mračna bodů. Vhodnějším řešením by bylo nalezení mediánu vzdáleností a odfiltrování všech korespondencí se vzdáleností menší než je hodnota nalezeného mediánu. Problém

takového řešení je časová náročnost nalezení mediánu, která je větší než při výpočtu průměru vzdáleností.

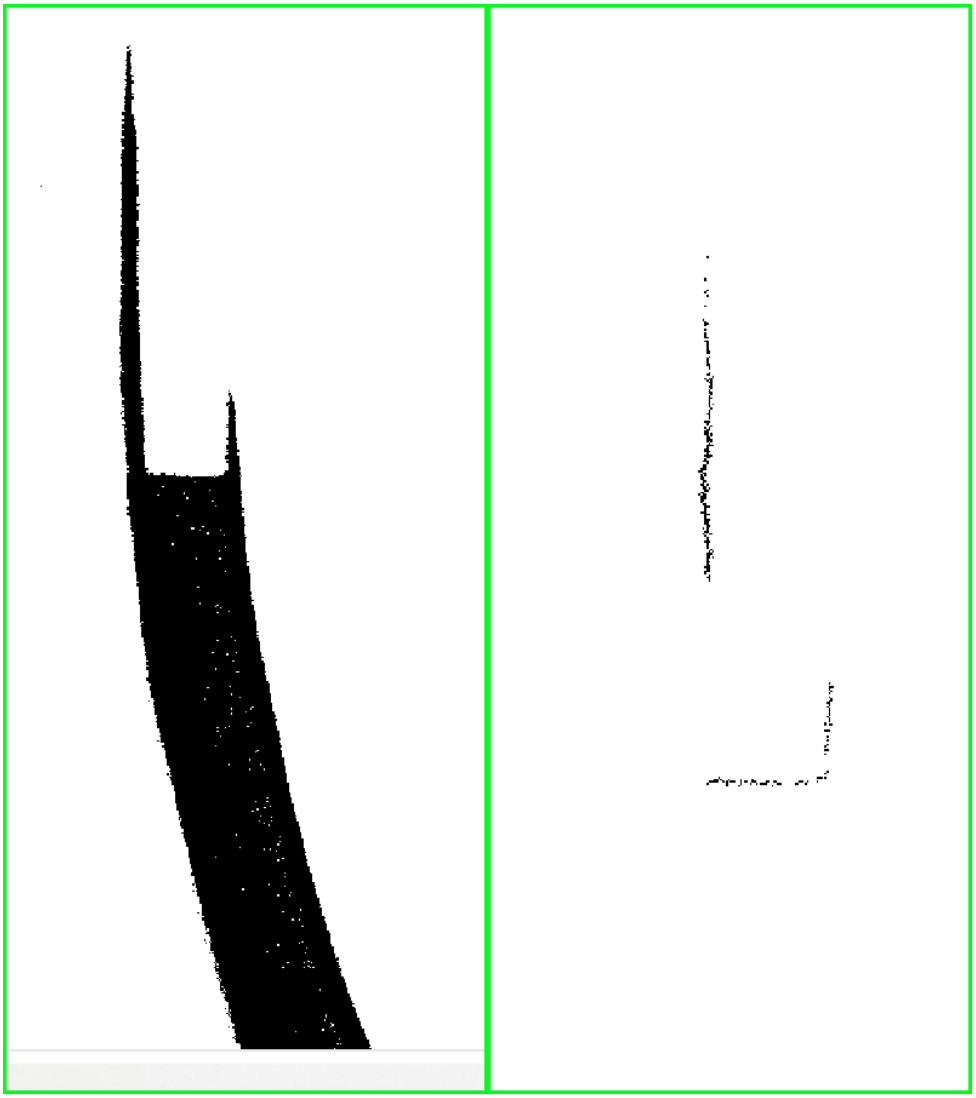
Ve většině případů se využívá více filtrů najednou, aby byly použity pouze nejvhodnější korespondence a výsledná transformace byla co nejpřesnější.

5.1.3 Použití implementace ICP v knihovně PCL k mapování a lokalizaci

Knihovna PCL [4] obsahuje třídu `IncrementalRegistraion`. Tato třída si ukládá vždy poslední mračno bodů. Toto mračno je potom použito jako cílové mračno. Problémem této třídy je skutečnost, že pokud algoritmus ICP skončí a zdrojové mračno je srovnané s mračnem cílovým, nalezená transformace je vynásobena s celkovou transformací (tato transformace by měla udávat pozici a natočení auta). To způsobovalo, hlavně když se auto nepohybovalo, nevhodné otáčení mračna které se stále navyšovalo s každým inkrementem.

To jsem vyřešil tak, že pokud není dosaženo určitého posunu mezi zdrojovým a cílovým mračnem, tak se nalezená transformace nevynásobí s celkovou transformací a ani cílové mračno není nahrazeno za aktuálně zpracovávané mračno. Tím se vyřešil problém se vznikem špatné rotace, když není auto v pohybu.

Na obrázku 16 je ozbrazen rozdíl mezi inkrementální registrací pomocí třídy `IncrementalRegistraion` a použitým řešením, když auto není v pohybu.



(a) Třída `IncrementalRegistraion` (b) Registrace až po určitém posunu

Obrázek 16: Inkrementální registrace

5.1.4 Vstupní mračna bodů

Z důvodu velké odchylky při mapování a určování pozice jsem zkoušel několik různých způsobů úprav mračen bodů před použitím v algoritmu ICP. Ze začátku jsem na skeny použil pouze funkci pro odstranění potenciálně chybných bodů z kapitoly 4. Toto řešení nepřinášelo moc dobré výsledky, hlavně z důvodu dlouhé doby běhu algoritmu ICP, což způsobovalo zahazování získaných skenů.

Další způsob, který jsem testoval bylo průměrování určitého počtu skenů. Zvolil jsem čtyři, protože čas výpočtu algortimu se pohyboval nad hodnotou 100 ms, a to značí čtyři skeny. Toto řešení nepřineslo žádné zlepšení oproti předchozí verzi.

Jelikož je knihovna PCL určena převážně pro zpracování 3D mračen bodů, otestoval jsem i možnost poskládat několik skenů s různou výškou (souřadnice z) přes sebe. Toto řešení se ukázalo jako nejvhodnější v kombinaci s voxelizací, viz kapitola 5.2, výsledného mračna kvůli zvětšení vzdálenosti mezi jednotlivými body, čímž se zlepšilo hledání korespondenčních bodů. Nevýhodou takového řešení je, že se zvětšuje počet bodů, které je nutné zpracovat, čímž se prodlužuje doba a počet vynechaných skenů.

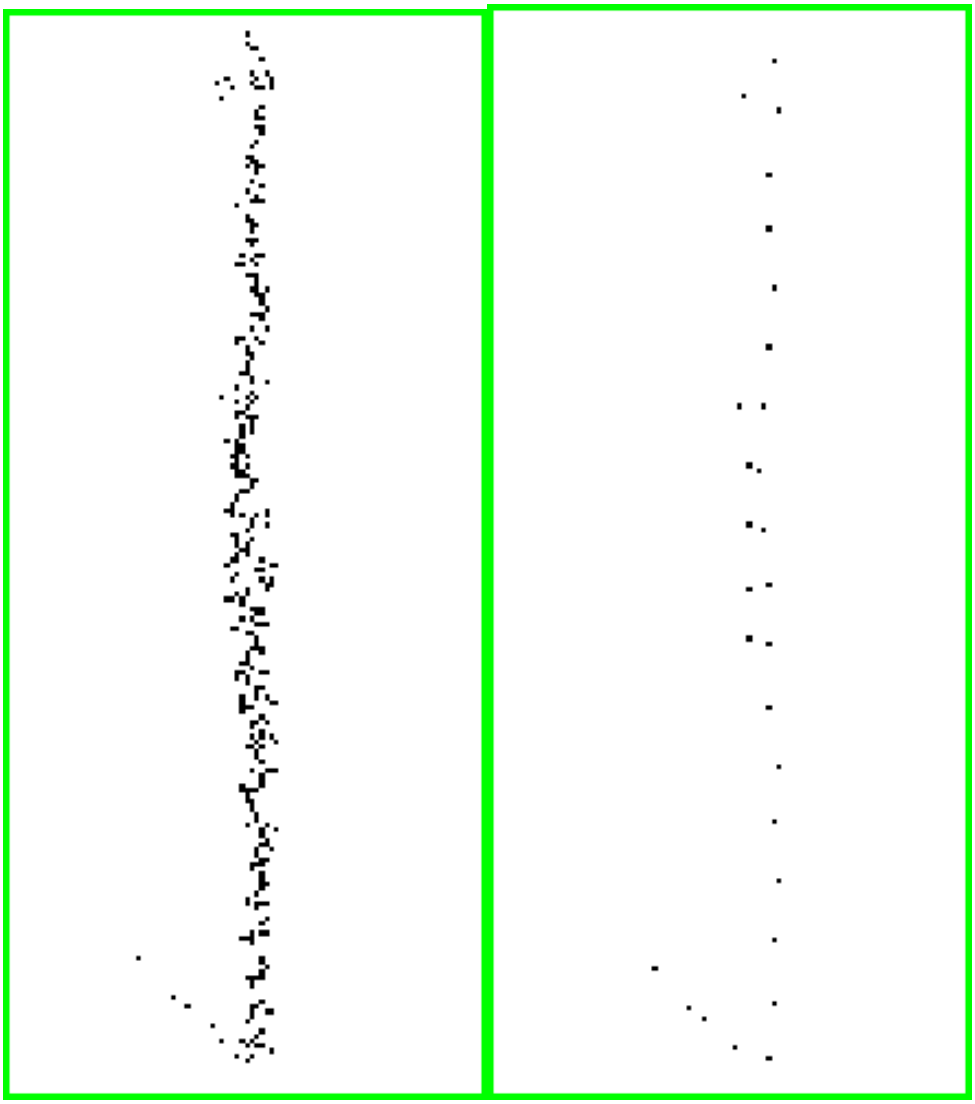
Nakonec jsem ještě tuto metodu upravil tak, aby byla vždycky nahrazena nejstarší vrstva. Toho jsem docílil tak, že jsem celé mračno posunul o dva centimetry níž (dva centimetry je výškový posun mezi vrstvami) a potom jsem odstranil všechny body, které mají výškovou hodnotu negativní. Tuto metodu jsem použil i při testech s výkonějším stolním počítačem který nahradil hlavní počítač.

5.2 Voxelizace mapy

K voxelizaci mapy se používá takzvaná 3D voxelová mřížka. Jedná se v podstatě o prostor kvádrů (voxelů) se zadanou velikostí, která je rozložena přes celé mračno bodů. V každém voxelu se potom aproximují jeho body s jejich těžištěm.

Při zvolení vhodné velikosti voxelu dojde ke znatelnému zmenšení mračna a zároveň nedojde k velké ztrátě rozlišení mapy.

Na obrázku 17 je vidět rozdíl před a po voxelizaci s velikostí voxelu 30x30mm. Reálně jsem použil voxelizaci s menší velikostí voxelu (pouze 5x5mm), čímž nedošlo k takové ztrátě bodů v mračnu. Mapa bude přesnější, ale budou odstraněny duplicitní body.



(a) Před použitím voxelizace (b) Po použitím voxelizace

Obrázek 17: Před a po použití voxelizace

5.3 Restartování pozice

V této práci není potřeba využití klasického řešení SLAM, jelikož se nejedná o plně autonomní vozidlo. Proto jsem se rozhodl, že po posunu o určitou vzdálenosti, kdy je auto řízeno ručně, se jeho pozice restartuje na souřadnice $[0, 0]$ a úhel natočení auta na $90°$.

Toho jsem dosáhl tak, že k rotační matici jsem spočítal matici inverzní. Poté je nutné touto maticí vynásobit negativní posun. Tím vznikne matice, která celou mapu posune tak, že aktuální pozice auta je nulová. Jelikož se jedná o 2D prostor je potřeba negativní posun otočit pouze o

`osu z`, k tomu slouží následující rovnice. Kde x' a y' je vynásobený negativní posun, α je úhel otočení a x a y je posun které auto urazilo od počátku souřadnic.

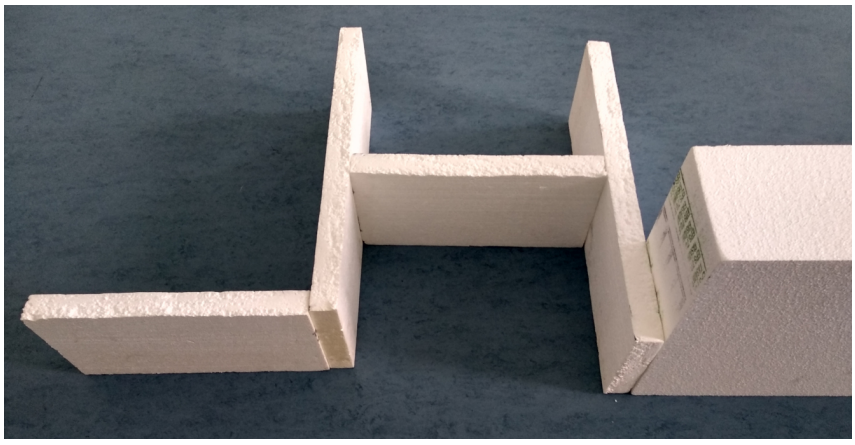
$$\begin{aligned} x' &= -xcos\alpha + ysin\alpha \\ y' &= -xsin\alpha - ycos\alpha \end{aligned} \tag{4}$$

Toto zjednodušuje použití filtru, který odstraní body, které jsou dál než v námi daném prostoru a hlavně hledání parkovacího místa. Takovýto filtr se nazývá CropBox. CropBox vytvoří nad vstupním mračnem kvádr definovaný dvěma body. Všechny body mračna, které jsou mimo tento kvádr, jsou odstraněny. Zjednodušení spočívá v tom, že není potřeba pokaždé přepočítávat hodnoty bodů, které nám určují velikost kvádru.

6 Nalezení parkovacího místa

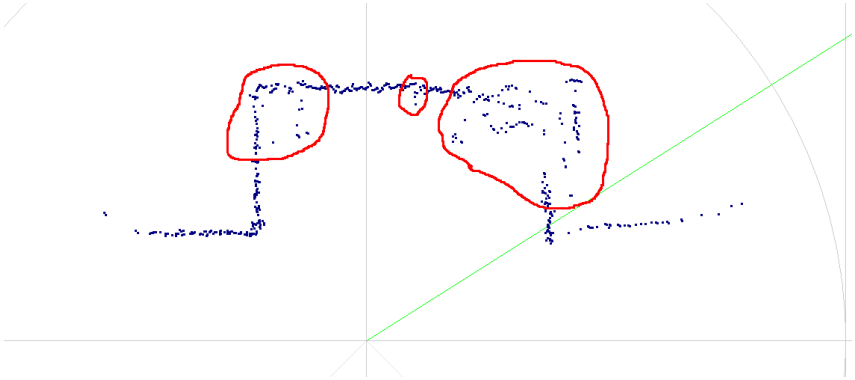
K simulaci parkovacího místa je v této práci vytvořen uměle ohraničený prostor o dostatečné velikosti, aby zde mohlo auto zaparkovat. Na obrázku 18 je parkovací místo ohraničené lepenkovými krabicemi. Tyto krabice se ukázaly jako nevhodné, protože z mně neznámého důvodu začaly po uběhnutí určité doby vytvářet chybná měření skenerem. Jakýkoliv jiný předmět, kterým krabice byly nahrazeny, tato chybná měření nevytvářel, dokonce ani, pokud byl vyroben z materiálu černé barvy, kterou skener detekuje nejhůře. Proto bylo parkovací místo ohraničeno bílým fasádním polystyrenem, který už problémy nezpůsoboval. Takto definované parkovací místo je na obrázku 19.

Obrázek 18: Prostor pro parkování vytvořený z lepenkových krabic



Obrázek 19: Prostor pro parkování vytvořený z fasádního polystyrenu

Na obrázku 20 jsou vidět chybná měření, která vznikali při skenování parkovacího místa z lepenkových krabic. Data byla čtena aplikací URG Viewer.



Obrázek 20: Chybná měření parkovacího prostoru z lepenkových krabic

Parkovací místo je definováno čtyřmi rohovými body. Pomocí těchto bodů je možné zjistit velikost a tvar parkovacího místa. Z těchto údajů jsme potom schopni zjistit, jestli je parkovací místo vhodné k parkování a pro jaký typ parkování je určeno. Parkovací místo může být určeno pro podélné parkování, kolmé parkování, pro oba způsoby parkování, anebo pro žádné parkování (tato možnost nastane, když parkovací místo není dostatečně velké ani pro jeden způsob parkování).

Pro detekci parkovacího místa je nejprve potřeba v mračnu bodů nalézt tři úsečky, které svírají úhel mezi 80° až 100° s `osou x`, jinak řečeno vertikální, a dvě úsečky, které svírají úhel 350° až 10° s `osou x`, jinak řečeno horizontální. Tyto úsečky reprezentují stěny parkovacího místa a průsečíky přímek, na kterých leží úsečky, jsou body, které definují parkovací místo.

6.1 Detekce přímek v mračnu bodů

Před detekcí úseček je nejvhodnější nalézt v mračnu bodů přímky a na těchto přímkách teprve hledat úsečky. Pro detekci přímek byla v této práci zvolena Houghova transformace. Houghova transformace je technika pro detekci vlastností používaná v analýze obrazu a strojovém vidění. Úkolem této techniky je vyhledávání pravidelných tvarů, jako jsou přímky, kruhy a elipsy. Původně byla určena právě jenom pro detekci přímek v obraze.

Houghova transformace využívá k detekci přímek výpočet vzdálenosti nejbližšího bodu na přímce sestrojené nad určitým bodem mračna. K tomu se využívá rovnice

$$r = xcos\Theta + ysin\Theta \tag{5}$$

Kde r je vzdálenost počátku souřadnicového systému k nejbližšímu bodu přímky, x a y je pozice bodu, kterým prochází přímka a Θ je úhel úsečky, který svírá vůči `ose x`, jež je sestavena mezi počátkem souřadnicového systému a bodem přímky. Ten je nejblíže k počátku souřadnicového systému. Tento úhel je také úhel, který svírá sestavená přímka vůči `ose y`. Pokud chceme tedy

z úhlu Θ udělat úhel, který přímka svírá vůči `ose x`, stačí k tomuto úhlu přičíst 90°. Pokud zde existuje přímka, bude mít několik bodů ve stejném úhlu Θ stejnou nebo podobnou hodnotu r.

V práci je Houghova transformace [10] implementována tak, že pro každý bod mračna se sestaví přímky s úhlem svírajícím s `osou x` v určitém rozsahu. Z těchto všech bodů se vybere jeden vzorový, se kterým se porovnávají vypočtené hodnoty ostatních bodů. Takhle se zjistí, kolik bodů se nachází na všech potenciálních přímkách a potenciální přímka s nejvíce body se považuje za správnou. Je nutné definovat minimální počet bodů, kterými musí přímka procházet. Dále je nutné určit nějaký rozptyl r, nebude totiž nikdy stejné u všech bodů z důvodu, že skener nedokáže dodat měření s tak velkou přesností. Určení rozptylu r je nutné určit podle přesnosti skeneru, když se zvolí rozptyl r moc malý, nebude docházet k detekci přímek z důvodu malého počtu bodů, přes které přímka prochází. Pokud se zvolí zase až moc velký, bude docházet ke zkreslování nalezených přímek hlavně v rozích, kde bude hledání ovlivněno body další přímky.

6.2 Detekce úseček v mračnu bodů

Po nalezení přímek v mračnech bodů je potřeba nalézt úsečky. V této práci jsem se rozhodl úsečky vyhledat tak, že zjistíme body, které leží na přímce. Tyto body jsem se rozhodl transformovat tak, aby přímka, kterou body tvoří, byla rovnoběžná s `osou x`. K tomu je potřeba zjistit úhel který je potom potřeba aplikovat na body přímky, čímž se zajistí, že tato přímka bude rovnoběžná s `osou x`. Tento úhel se dá spočítat odečtem úhlu, který přímka s `osou x` od 360°.

Úsečka je reprezentována krajními body A a B. Tyto body je potřeba najít. Nejdříve jsou tyto body nastaveny na vzorový bod, který byl vybrán při detekci přímek. Poté se hledá bod, který se na `ose x` nachází více vlevo, neboli má souřadnici x menší než bod A. Zároveň je potřeba, aby tento bod neměl souřadnici x menší o velký rozdíl, jinak by nedošlo k detekci úseček, ale pouze k detekci dvou bodů na přímce, které jsou od sebe nejvíce vzdálené, přičemž na jedné přímce se může nacházet více úseček. To samé je potřeba provést i s bodem B, jenom s tím rozdílem, že se zde hledá bod, který je na `ose x` více vpravo, neboli má hodnotu x větší než bod B. Tato operace se iteruje tak dlouho, dokud při iteracích dochází ke změně jednoho z bodů, pokud již nebyl ani jeden bod změněn. Znamená to, že byla nalezena úsečka.

Aby nalezená úsečka byla co nejlépe umístěna na přímce, na které leží, byla vypočítána také průměrná souřadnice y ze všech bodů, které leží na dané úsečce. Tato průměrná hodnota souřadnice y byla zapsána jako souřadnice y krajních bodů A a B. Jelikož všechny body, které leží na přímce, byly transformovány tak, aby tato přímka byla rovnoběžná s osou x, jsou i body A a B transformovány a úsečka kterou reprezentují je také rovnoběžná s `osou x`. Tyto body je tedy nutné otočit podle `osy z` o úhel, který původní přímka svírala s `osou x`.

6.3 Spojování úseček

Při vyhledávání přímek v mračnu bodů reprezentujícího parkovací místo často docházelo k tomu, že jedna stěna parkovacího místa byla složena z více úseček a tyto úsečky bylo potřeba složit

tak, aby každá úsečka tvořila jednu stěnu parkovacího místa, aby bylo možné spočítat průsečíky těchto přímek, na kterých úsečky leží a tím získat rohové body parkovacího místa.

V této práci jsem se nejdříve rozhodl úsečky seřadit. Vertikální úsečky se řadí podle souřadnice x středového bodu úsečky. U horizontálních úseček probíhá řazení podle vzdálenosti od počátku souřadnicového systému.

Potom je možné jednoduše tyto úsečky spojit aby tvořily úsečky, které reprezentují stěny parkovacího místa. Toho je dosaženo tak, že má bod B jedné úsečky a bod A druhé úsečky podobnou souřadnici x u vertikálních úseček a souřadnici y horizontálních úseček. Tyto úsečky je možné spojit do jedné. Pokud nejsou hodnoty souřadnic podobné, jedná se o začátek jiné úsečky.

6.4 Výpočet rohových bodů parkovacího místa

Pokud ve výsledku zbyly tři vertikální a dvě horizontální úsečky, je pravděpodobné, že zde byl nalezen prostor připomínající parkovací místo. K definování parkovacího místa je potřeba zjistit jeho čtyři krajní body. V této práci jsou rohové body určeny výpočtem průsečíku dvou přímek. Na těchto přímkách leží úsečky identifikující stěny parkovacího místa.

Pro výpočet průsečíku dvou přímek je potřeba znát obecnou rovnici přímek. K zjištění obecné rovnice přímky je potřeba znát normálový vektor přímky. Ten lze získat ze směrového vektoru přímky. K výpočtu směrového vektoru je potřeba znát dva body ležící na této přímce. Těmito body jsou body určující úsečku, která leží na této přímce. K výpočtu vektoru lze použít tento vzorec:

$$u = (x_A - x_B, y_A - y_B) \tag{6}$$

Pomocí směrového vektoru lze získat normálový vektor přímky. Tento vektor lze získat prohozením souřadnic a převodu jedné souřadnice na zápornou. Výsledná výpočet bude vypadat takto:

$$n = (u_2, -u_1) \tag{7}$$

Nebo s opačným směrem:

$$n = (-u_2, u_1) \tag{8}$$

Normálový vektor lze dosadit do obecné rovnice přímky společně s jedním bodem, který leží na přímce. Po dosazení vznikne rovnice:

$$n_1 x_A + n_2 y_A + c = 0 \tag{9}$$

Jedinou neznámou rovnice je nyní c. Z rovnice je nutno vyjádřit c. Po všech úpravách bude mít rovnice tvar:

$$c = -n_1 x_A - n_2 y_A \tag{10}$$

Výsledná obecná rovnice přímky bude ve tvaru:

$$ax + by + c = 0 \tag{11}$$

Kde a, b, c jsou konstanty, které jsme vypočítali předešlými kroky a x, y jsou neznámé. Když máme takto vyjádřené rovnice dvou přímek, na kterých leží úsečky, které odpovídají stěnám parkovacího místa, je z rovnic přímek potřeba vyjádřit průsečík těchto přímek. Jedná se o soustavu dvou rovnic o dvou neznámých.

$$\begin{aligned} ax + by + c &= 0 \\ ex + fy + g &= 0 \end{aligned} \tag{12}$$

V těchto dvou rovnicích jsou a, b, c, e, f, g konstanty a x, y jsou souřadnice našeho průsečíku. Nejdříve je tedy z jedné z rovnic potřeba vyjádřit x nebo y a toto vyjádření dosadit do rovnice druhé. Vyjádření x a následné dosazení bude vypadat takto:

$$\begin{aligned} x &= \frac{-by - c}{a} \\ y &= \frac{\frac{ce}{a} - g}{\frac{-be}{a} + f} \end{aligned} \tag{13}$$

Problémem toho výpočtu je, že pokud je a nulové tak dojde k dělení nulou a to způsobí nenalezení průsečíku, výsledné souřadnice jsou rovny hodnotě NAN. Pokud je teda a nulové tak se z první rovnice vyjádří y a to je potom dosazeno do druhé rovnice. Výsledná úprava bude ve tvaru:

$$\begin{aligned} y &= \frac{-0x - c}{b} \\ x &= \frac{-f\frac{-c}{b} - g}{e} \end{aligned} \tag{14}$$

Pokud i tento výpočet vrátí průsečík s hodnotou souřadnic rovnu NaN. Znamená to, že přímky jsou rovnoběžné a to znamená, že nemůže existovat jejich průsečík.

7 Parkování

Pro kolmé i podélné parkování jsem použil algortimy z diplomové práce Automatické parkování automobilu od Luboše Matejčíka [1], rozhodl jsem se proto, že algoritmy vykazovaly vysokou úspěšnost za různých podmínek, které při kolmém parkování dosahovaly průměrně 93% úspěšnosti a při podélném parkování 76% úspěšnosti. Algoritmy bylo nutné upravit tak, aby využívali prvky knihovny PCL. Bylo tedy nutné nahradit použití třídy $Point2$ za $PointXYZ$ z knihovny PCL a také nahradit třídu $Vector2$ taktéž za $PointXYZ$ z knihovny PCL.

Dále bylo potřeba vytvořit několik metod pro práci s body, které knihovna neobsahuje, jako je například dělení a násobení bodu konstantou, normalizace vektoru.

8 Výsledky navrženého řešení

Navržené řešení není schopno autonomně zaparkovat. Hlavním důvodem je nízký výpočetní výkon hlavního počítače. Čas výpočtu algoritmu ICP je moc vysoký mezi 100 až 250 ms, to způsobuje přeskakování skenů (4 až 10). Při nejnižší možné rychlosti modelu auta hlavní počítač není schopen vytvářet mapu okolí a tím pádem není schopen ani určovat svoji pozici. Pokud je model auta posouván rukou velmi nízkou rychlostí, tak je možné určit přibližnou pozici auta a vytvořit přibližnou mapu okolí, ale chybovost se stále pohybuje až v jednotkách centimetrů.

Řešení bylo testováno i na výkonnějším počítači. K testům byl použit notebook s procesorem Intel Core i7 6500U s frekvencí 3,1 GHz, dvěma jádry s podporou čtyř vláken, 8 GB operační paměti a 240 GB SSD diskem. Na notebooku je nainstalován operační systém Windows 10 Pro x64, ale pro testovací účely byl v programu Oracle VM VirtualBox vytvořen virtualizovaný stroj s přiřazeným čtyřmi vlákny, 4 GB operační paměti a 20 GB pevným diskem. Na tomto virtualizovaném stroji je nainstalován Linux Ubuntu 18.04. Na tomto stroji byl čas výpočtu algoritmu ICP mezi 20 až 35 ms, takže docházelo ke ztrátě maximálně jednoho skenu. Při tomto testu bylo již možné vytvářet mapu okolí za jízdy modelu auta. Ale nevyřešilo to problém s kvalitou mapy a odchylka se stále pohybovala v jednotkách centimetrů, což stále stěžuje detekci parkovacího místa a určení pozice.

Jedním z možných řešení, které by řešilo problém s dobou běhu ICP algoritmu je rozdělení algoritmu na více vláken tak, aby běžely paralelně vedle sebe. Další možností, která by zrychlila běh algoritmu, ale i jeho přesnost by bylo získání přesnější předpokládané transformace. Toho se dá dosáhnout několika způsoby například pomocí měření otáček kol, tím odhadovat předpokládaný posun modelu auta a ICP algoritmus by sloužil jen k zpřesnění tohoto měření, hlavně pokud by docházelo k prokluzu kol, anebo využit algoritmu pro odhad překládaného posunu. U těchto algoritmů je problém, že většinou jsou tyto algoritmy založeny na vlastnostech bodů tzv. feature based algorithms, použitím dalšího algoritmu by se, ale zvýšila náročnost celého programu.

Jedním z možných řešení by mohlo být také použití jiné knihovny než knihovny PCL, která je primárně určená pro 3D mračna bodů, i když obsahuje některé třídy i pro práci s 2D mračny. Jednou z možných knihoven, která by mohla být použita je knihovna MRPT. Tato knihovna byla použita i v diplomové práci [1] ze, které tato práce vychází. Jedním z vhodných vylepšení by bylo využít zjištěný odběr proudu motoru pro plynulejší rozjezd auta i při nižším výkonu, vytvořením proudové špičky při rozjezdu.

9 Závěr

Cílem práce bylo vytvořit parkovacího asistenta pro kolmé i podélné parkování, který detekuje parkovací místo a následně na něj zaparkuje. Vytvořit funkčního asistenta, se povedlo pouze částečně. Parkovací asistent je schopný detekovat parkovací místo a určit svoji pozici, ale s neuspokojivými výsledky. Jedním z důvodů je velká odchylka měření použitého laserového skeneru Hokuyo UST-20LX, která se pohybuje kolem 20 mm, v případě rozměrů použitého modelu auta je to asi 5 až 10 %, což ovlivňuje lokalizaci a mapování, později pak nalezení parkovacího místa a následné parkování. V případě přesnějšího měření by měl model auta být schopen automaticky zaparkovat.

Ze začátku jsem seznámil s diplomovou prací Automatické parkování automobilu [1] a navrhoval model automobilu, rozmístění a uchycení součástek. Poté jsem se seznámil s komunikací se skenerem pomocí protokolu SCIP [19], navrhoval třídu pro komunikaci se skenerem a program využívající gamepad pro řízení auta . Později jsem zjišťoval možnosti knihovny PCL a její použití pro zpracování bodů hlavně při lokalizaci a mapování za použití algoritmu ICP a možnosti využití při vyhledávaní parkovacího místa. Nakonec jsem upravoval algoritmy pro podélné a kolmé parkování z výše uvedené diplomové práce a zlepšování činnosti algoritmu ICP aby jej bylo možné využít pro lokalizaci a mapování. Vyzkoušel jsem několik různých metod pro předzpracování mračen bodů např. skládání několika mračen přes sebe, aby byl vytvořen 3D prostor, průměrování několika po sobě jdoucích skenů nebo různé algoritmy pro odstranění chybných bodů. Některé tyto metody přispěly ke zlepšení průběhu lokalizace a mapování.

Práci stěžovala doba překladu navrženého programu, který trval po každé změně minimálně minutu a při překladu celého programu asi osm minut, což komplikovalo testování navrženého řešení. Překlad některých knihoven bylo potřeba provést dvakrát z důvodu, že při prvním překladu knihovny PCL došlo k vyzužití veškeré operační paměti a pádu překladu knihovny. Systém jsem následně znovu nainstaloval a na SSD disku vytvořil SWAP oddíl. Poté jsem překlad provedl znovu tentokrát bez problému.

Literatura

[1] MATEJČÍK, Luboš. *Automatické parkování automobilu* [online]. Ostrava, 2017 [cit. 2019-04-21]. Dostupné z: http://hdl.handle.net/10084/119004. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava.

[2] MIKULČÁK, Jiří. *Matematické, fyzikální a chemické tabulky a vzorce pro střední školy*. Praha: Prometheus, 2003. ISBN 978-80-7196-264-9.

[3] *URG Network* [online]. [cit. 2019-04-21]. Dostupné z: https://sourceforge.net/projects/urgnetwork/

[4] *PCL* [online]. [cit. 2019-04-21]. Dostupné z: http://pointclouds.org/

[5] *Laser rangefinder*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-04-21]. Dostupné z: https://en.wikipedia.org/wiki/Laser_rangefinder

[6] *Lidar*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-04-21]. Dostupné z: https://en.wikipedia.org/wiki/Lidar

[7] *Iterative closest point*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-04-21]. Dostupné z: https://en.wikipedia.org/wiki/Iterative_closest_point

[8] *Simultaneous localization and mapping*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-04-21]. Dostupné z: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

[9] *Ackermannova podmínka*. Autolexicon.net [online]. [cit. 2019-04-21]. Dostupné z: http://www.autolexicon.net/cs/articles/ackermannova-podminka/

[10] *Hough transform*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-04-21]. Dostupné z: https://en.wikipedia.org/wiki/Hough_transform

[11] *I.MX 6Dual/6Quad Applications Processor Reference Manual* [online]. [cit. 2019-04-24]. Dostupné z: https://www.nxp.com/webapp/Download?colCode=IMX6DQRM

[12] *FRDM-K64F: Freedom Development Platform for Kinetis® K64, K63, and K24 MCUs* [online]. [cit. 2019-04-24]. Dostupné z: https://www.nxp.com/support/developer-resources/evaluation-and-development-boards/freedom-development-boards/mcu-boards/freedom-development-platform-for-kinetis-k64-k63-and-k24-mcus:FRDM-K64F

[13] *SolidRun Wiki* [online]. [cit. 2019-04-24]. Dostupné z: https://wiki.solid-run.com

[14] *Eigen* [online]. [cit. 2019-04-24]. Dostupné z: http://eigen.tuxfamily.org

[15] *FLANN* [online]. [cit. 2019-04-24]. Dostupné z: https://www.cs.ubc.ca/research/flann/

[16] *Boost* [online]. [cit. 2019-04-24]. Dostupné z: https://www.boost.org/

[17] *Hokuyo UST-20LX* [online]. [cit. 2019-04-24]. Dostupné z: https://www.hokuyo-aut.jp/search/single.php?serial=167

[18] *OpenCV* [online]. [cit. 2019-04-24]. Dostupné z: https://opencv.org/

[19] *SCIP* [online]. [cit. 2019-04-27]. Dostupné z: https://www.generationrobots.com/media/Hokuyo%20UST-10LX/UST-10LX_20LX_SCIP_specifications_C-42-04076.pdf

[20] *Freescale Cup Shield for the Freedom KL25Z* [online]. [cit. 2019-04-27]. Dostupné z: https://community.nxp.com/docs/DOC-93914

[21] *Polární soustava souřadnic*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-04-28]. Dostupné z: https://cs.wikipedia.org/wiki/Pol%C3%A1rn%C3%AD_soustava_sou%C5%99adnic

[22] *Bezpečnost laseru Třídy 1 až 4* [online]. [cit. 2019-04-29]. Dostupné z: http://www.lt.cz/e-learning/laser/bezpecnost-laseru-tridy-1-az-4

A Výpis elektronické přílohy

/Latex/
    main.tex
/Literatura/
    UST-10LX_20LX_SCIP_specifications_C-42-04076.pdf
    IMX6DQRM.pdf
    MAT0137_FEI_N2647_2612T025_2017.pdf
/OpenSCAD/
    holder_alamak_bat.scad `//Model držáku baterie`
    holder_FRDM_K64F.scad `//Model držáku K64F`
    holder_HummingBoard.scad `//Model držáku pro hlavní počítač`
    holder_lrf_ust20lx.scad `//Model držáku pro Hokuyo UST-20LX`
    holder_dc-dc.scad `//Model držáku DC/DC měniče`
    holder_FET_switch.scad `//Model držáku zapínání skeneru`
/Pouzite_Obrazky/
    ackerman.png `//Náčrtek použitého podvozku`
    car.jpg `//Fotografie modelu auta`
    connection.png `//Blokové schéma zapojení modelu auta`
    err_scan.png `//Chybný sken při požití lepenkových krabic`
    filter_a.png `//Před použitím filtru pro odstranění chybných bodů v mračnu`
    filter_b.png `//Po použitím filtru pro odstranění chybných bodů v mračnu`
    Forest_LIDAR.jpg `//Snímek pořízený metodou LIDAR`
    gd.png `//Popis příkazu GD`
    hummingboard_bottom.jpg `//Fotografie hlavního počítače`
    hummingboard_holder.png `//Model držáku hlavního počítače`
    hummingboard_top.jpg `//Fotografie hlavního počítače`
    Incremental1.png `//Inkrementální registrace 1`
    Incremental2.png `//Inkrementální registrace 2`
    ip.png `//Popis příkazu IP`
    k64f.jpg `//Fotografie FRDM-K64F`
    k64f_holder.png `//Model držáku FRDM-K64F`
    md.png `//Popis příkazu MD`
    parking_place_1.jpg `//Parkovací místo z lepenkových krabic`
    parking_place_2.jpg `//Parkovací místo z fasádního polystyrenu`
    polar.png `//Převod mezi polární a kartézskou soustavou souřadnic`
    scanner.jpg `//Hokuyo UST-20LX`
    tfc.jpg `//Klon FRDM-TFC`
    voxel_a.png `//Před použitím voxelizace`

voxel_b.png `//Po použitím voxelizace`

/Text_prace/

2019_HAL0158_BP.pdf

/Zdrojove_kody/

BScanner.cpp

BScanner.h

Car.cpp

Car.h

CMakeLists.txt

Joystick.cpp

Joystick.h

Line.cpp

Line.h

LineDetection.cpp

LineDetection.h

main.cpp

main.h

Map.cpp

Map.h

MyCorrespondenceRejector.cpp

MyCorrespondenceRejector.h

ParkingPlace.cpp

ParkingPlace.h

ParkingPlaceDetection.cpp

ParkingPlaceDetection.h

PointMethods.cpp

PointMethods.h

Scanner.cpp

Scanner.h

TFC.cpp

TFC.h

TFCSerial.cpp

TFCSerial.h